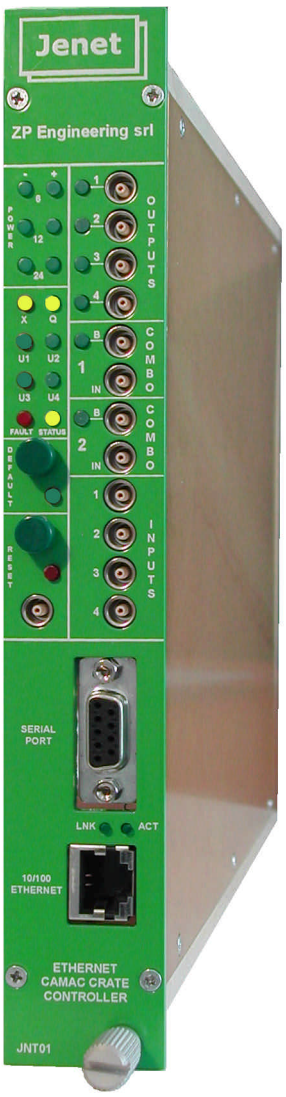



# JENET

## Ethernet CAMAC Crate Controller

www.zpeng.com/jenet

# User's Manual

P/N: **JNT01**
Ver: 1.0.2
Date: 25 Feb 2004

ZP Engineering srl
Via Michelangelo Peroglio, 15
00144 Rome, ITALY
www.zpeng.com

**INDEX**

1 General information ..... 4
1.1 Warranty ..... 4
1.2 System overview ..... 5
1.3 Summary of features ..... 5
2 Starting up ..... 6
3 Serial port control ..... 7
4 Startup options ..... 7
4.1 Startup flags ..... 7
4.2 Jumper settings ..... 8
5 Remote control ..... 9
5.1 TCP control socket ..... 9
5.2 ESONE library ..... 10
6 Local web server ..... 11
6.1 Commands page ..... 12
6.2 NIM I/O page ..... 13
6.2.1 Input section ..... 13
6.2.2 Output section ..... 14
6.2.3 COMBO section ..... 14
6.3 System settings page ..... 15
6.4 Diagnostics page ..... 16
7 Local scripting ..... 17
7.1 Lua scripting language ..... 17
7.2 Lua engine in Jenet ..... 17
7.2.1 Bit manipulation extension ..... 18
7.2.2 Socket commands for Lua control ..... 18
7.2.3 Jenet Script Manager (jsm) ..... 19
7.2.4 Scripting on Jenet ..... 19
8 Firmware Upgrade ..... 20
9 NIM subsection ..... 20
9.1 Default button ..... 20
9.2 Inputs ..... 21
9.3 Outputs ..... 22
9.4 COMBO I/O ..... 23
10 Commands reference ..... 24
11 System Specifications ..... 28

*This log identifies those portions of this document, which have been revised since the original issue and the date of each revision.*

| Rev | Summary of changes to previous version | Date | Approval |
|---|---|---|---|
| 1.0 | First version | 15/01/04 | UZ, PP, MV, FP |
| 1.0.1 | Notes added to sections 3, 5.1 and 8. | 11/02/04 | UZ |
| 1.0.2 | Lua binary operators added (section 7.2.1 and section 10)<br>Control socket commands for Lua control added (section 7.2.2)<br>Revised jsm description (section 7.2.3) | 24/02/04 | UZ |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# 1 General information

This manual is intended to provide a reference for operating and programming JENET, an Ethernet-based CAMAC crate controller.

## *1.1 Warranty*

**HARDWARE**

ZP Engineering srl warrants the JENET JNT01 product to operate within specifications under normal use and service for a period of 2 (two) years from the date of shipment. Component products, replacement parts and repairs are warranted for 90 days. This warranty extends only to the original purchaser.

Should the product, in ZP Engineering's opinion, malfunction within the warranty period, ZP Engineering will at its discretion repair or replace the product upon receipt with an equivalent. Any replaced parts become property of ZP Engineering. This warranty does not apply to a product that has been damaged due to accident, misuse, abuse, improper installation, usage not in accordance with product specifications and instructions, natural or personal disaster, or unauthorized alterations, repairs or modifications.

Please note the following:

- The warranty applies only to defects in workmanship.
- The warranty makes no claims regarding suitability of the product for a particular use or environment.
- A serial number is required for any warranty service.
- Proof of purchase may be required; ZP Engineering srl accepts originals, photocopies and faxes as proof of purchase when required.
- Unauthorized repairs to a ZP Engineering board level product will void the warranty offered by ZP Engineering srl.
- ZP Engineering srl reserves the right to refuse to service any product that has been altered, modified or repaired by non-ZP service personnel.
- Standard warranty service consists of repair upon receipt. You are responsible for the cost of shipping the product to ZP. ZP pays the cost of returning a product to you.
- ZP Engineering srl reserves the right to replace the product with a service product at their sole discretion at any time.
- This warranty does not apply to the software component of a product.
- Components and schematics are property of ZP Engineering srl and are not available to end users.

**SOFTWARE**

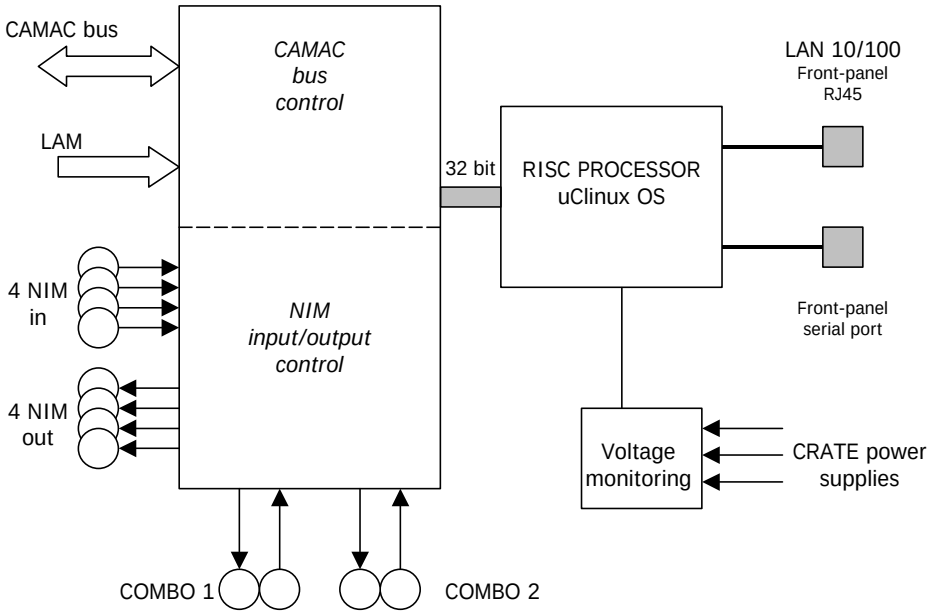
Software included in the unit has been carefully tested, but is supplied “as is” with no warranty covering detailed performance. Critical problems are covered by a software warranty for a period of 3 (three) months from the date of shipment. Within this period, software upgrades will be provided; software upgrades will be performed by the customer using the available tools.

**DOCUMENTATION**

The documentation included with the unit represents the latest version document, and is continually refined end edited. Please refer to the support web site at www.zpeng.com/jenet for the latest version.

# 1.2 System overview

JENET is a complete CAMAC controller that allows advanced interaction by means of standard Ethernet services, such as a local web server and TCP socket based communication protocol. The basic hardware architecture of JENET is illustrated below.



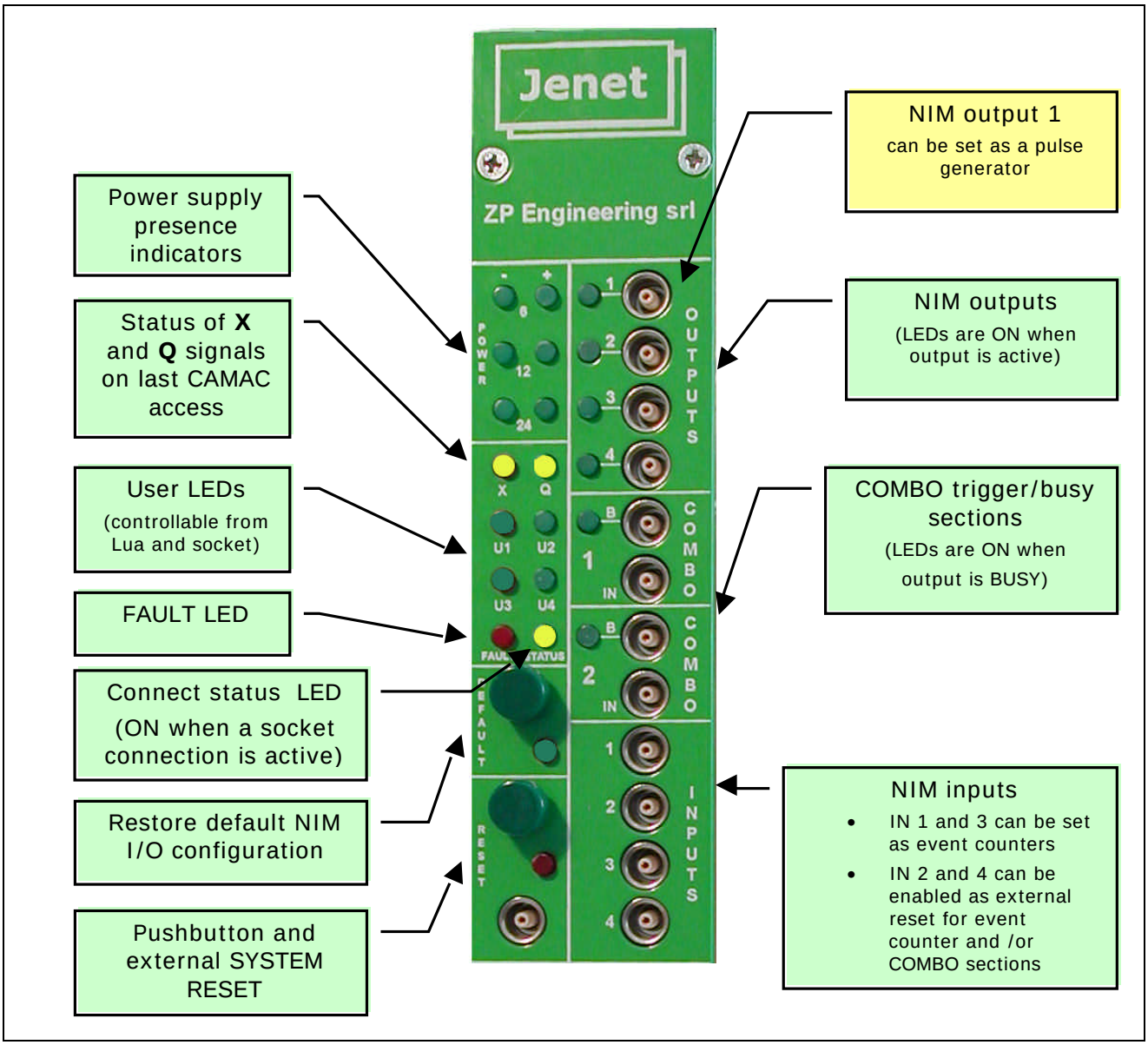

A local processor runs a version of Linux optimized for low memory footprint; a CAMAC bus control subsection handles all bus access operations and interactions, and a separate NIM subsection manages I/O signals located on the front panel.

# 1.3 Summary of features

| | |
|---|---|
| **CAMAC bus access** | o Full CAMAC bus control, including LAM detection<br>o Plugs into slots 24 and 25 |
| **Local NIM I/O Section** | o 4 outputs, 4 inputs, event counters, 2 COMBO I/O (trigger/busy) modules<br>o programmable pulse generators<br>o input event counters<br>o NIM default settings can be reloaded with front panel button |
| **Remote Control Library** | o ANSI C remote control library derived from the ESONE standard, with extensions to control local resources<br>o Remote control of all functions through TCP socket |
| **Local Web Server** | o Dynamic local web server allows advanced monitoring and control without the need to install dedicated software (perfect for crate setup and maintenance)<br>o User page with results from script |
| **Advanced Scripting Engine** | o Embedded script interpreter allows local execution of C-like code, with full control on CAMAC and NIM functions<br>o No need to install cross-compilation toolchains |
| **Front Panel Indicators** | o X and Q signals on last access<br>o 4 user LEDs (controllable from script)<br>o Fault, connection status and NIM default indicators |

# 2 Starting up

1. Please insert the controller **into slots 24 and 25 ONLY** of a standard CAMAC crate (rightmost when looking the crate from the front side)
2. if default network settings are compatible with your setup, connect a LAN cable to the front-panel RJ45 socket
3. Power up the CAMAC crate
4. Wait about 20 seconds to allow completion of operating system boot
5. If default network settings are not suitable for your network environment, connect a terminal to the front panel serial port and make the necessary variations (see **Serial port control** section). After that, reboot (either by cycling the main power or by pressing the RESET front panel button) and wait about 20 seconds.
6. For instant gratification, please open your favorite browser (on a host connected to the same LAN used by Jenet) and point to its IP address; if default settings apply, the default IP address is 192.168.0.98. From the werb server pages, you already have control over the NIM I/O section and the possibility to perform individual commands on the CAMAC bus.


Jenet
ZP Engineering srl
POWER
OUTPUTS
COMBO
INPUTS
DEFAULT
RESET
X
Q
U1
U2
U3
U4
Power supply presence indicators
Status of X and Q signals on last CAMAC access
User LEDs (controllable from Lua and socket)
FAULT LED
Connect status LED (ON when a socket connection is active)
Restore default NIM I/O configuration
Pushbutton and external SYSTEM RESET
NIM output 1 can be set as a pulse generator
NIM outputs (LEDs are ON when output is active)
COMBO trigger/busy sections (LEDs are ON when output is BUSY)
NIM inputs
IN 1 and 3 can be set as event counters
IN 2 and 4 can be enabled as external reset for event counter and /or COMBO sections

# 3 Serial port control

A serial port connector located on the front panel allows the user to modify system settings; this procedure is required if current network parameters are incompatible with the local network. It is also possible to modify startup options.

Connection parameters are the following: 38400 baud, 8-N-1, no flow control. Echo is not enabled on the serial port, so please enable character echo on your serial terminal.

The following commands are available:

| | |
|---|---|
| help | Provides a quick list of commands on terminal |
| setip <new IP addr> | sets a new IP address, to be written in the format *aaa.bbb.ccc.ddd* |
| setmask <new mask> | sets a new IP mask |
| setgw <new gw addr> | sets a new gateway IP address |
| setdhcp <0\|1> | if set to 1, enables the local DHCP client |
| getip<br>getmask<br>getgw<br>getdhcp | Allows retrieval of current network settings |
| setrob <0\|1> | if set to 1, enables the Lua Run-On-Boot option |
| setcscan <0\|1> | if set to 1, enables the Crate Scan function (executed at startup only) |
| getrob<br>getcscan | Allows retrieval of current startup settings (Lua Run-On-Boot and Crate Scan) |

| | |
|---|---|
| *Note:* | *The front-panel connector requires a straight serial cable (pin 2 to pin 2, pin 3 to pin 3); only RX and TX are required.* |

# 4 Startup options

Jenet startup options can be tuned in order to provide a fine control over performance and connectivity. There are some settings stored in nonvolatile memory that can be changed with the control socket or from the serial port, and some options that require placement or removal of jumpers inside the unit.

## *4.1 Startup flags*

Two flags are available for startup fine-tuning:

- Lua script Run-On-Boot flag
- Crate Scan Enable flag

Both are stored in the local EEPROM and can be modified with dedicated commands on the serial port (**getrob**, **setrob**, **getcscan**, **setcscan**) or from the control socket.

The **Lua Run-On-Boot** flag, when enabled, tells the system to run the stored Lua script after starting up the application; its main usage is for automated crate initialization and for unattended control.

**Crate Scan** is a function available on Jenet to allow automatic detection of cards inserted into the crate. It is a quick way of verifying the presence of cards that may be required by the acquisition code (either within the Lua script or on the host application).

Being a heuristic approach to card detection (there is no formal way of detecting a card when inserted into the crate) the Crate Scan function may interfere with specific CAMAC cards; it is thus possible to disable the Crate Scan function. Note that Crate Scan is executed only at startup (in order to avoid possible interaction with ongoing script or actions from host); if the feature is disabled, it will not be possible even from socket.

A description in pseudo-code of the Crate Scan function follows:

```
for (slot=1; slot < 23; slot++)     {
   end_slot = 0
   for (fun=0; fun < sizeof(SCAN_FUNCTION); fun ++)
      for (addr = 0; addr < 32; addr ++)   {
         X = CSSA (slot, SCAN_FUNCTION[fun],addr, 0)
         if (X == 1)  {
                  slot_status[slot] = 1
                  end_slot = 1
                  }
         }
   if (!end_slot)  slot_status[slot] = 0
   }
```

*where SCAN_FUNCTION is the following array:*

```
0, 1, 2, 3, 8, 9, 10, 11, 24, 25, 26, 27, 16, 17, 18, 19
```

# *4.2 Jumper settings*

It is possible to force some startup options by placing or removing internal jumpers.

Only experienced personnel should perform this operation.

The operation is described in steps, as follows:

- switch off power from the crate
- remove the controller from the crate
- remove the lateral cover (left side when looking on the front panel) of the controller; you should see the internal boards with components facing your side
- locate the jumper block, right behind the serial port connector
- note that if all jumpers are removed (default condition), then the unit will perform in the standard mode; insert jumpers only if you want to modify the standard setup, according to the table:

| Insert jumper on | to disable | Notes |
|---|---|---|
| **JMP10** | TELNET | Disable telnet to avoid system access to local resources; used only for performing a firmware upgrade or for debug. |
| **JMP9** | WEB SERVER | Disable web server if HTTP access not required in order to obtain maximum performance; note that it cannot run if application is also disabled |
| **JMP8** | APPLICATION | Disable application if the system hangs after powerup (i.e., after failing a firmware upgrade) |
| **JMP7** | COM SERVER | Disable serial port application if the system keeps hanging after powerup; it may also be used to prevent local access to critical parameters (i.e., network settings) |

# 5 Remote control

The unit has been designed to allow full control from a remote location, taking advantage of the available high-speed Ethernet interface.

A summary of the implemented TCP/IP services is detailed.

| Service | Port | Notes |
|---|---|---|
| HTTP server | 80 | Dynamic Web Server; it serves up to N client browsers at the same time. |
| TCP server | 2000 | TCP control socket; up to N different clients are allowed at the same time. |
| TCP client | -- | Connects to host TCP server for interrupt management only. |
| Telnet server | 23 | System telnet server; it is used mainly for firmware updates and may be disabled by the user (see **Startup options** section).<br>Please note that the telnet connection is unsafe (no password-protected access, no encryption). |

The internal software architecture is designed to allow control of multiple crates. It must be considered that, when using an Ethernet-based CAMAC controller, the distinction between crates is implicit as every crate is identified by a different IP address. Therefore the crate number is typically not a parameter in many command definitions.

The local web server allows quick testing and monitoring of the CAMAC crate, while the socket connection is the main control method for general applications; the availability of an internal scripting engine represents an additional plus.

## *5.1 TCP control socket*

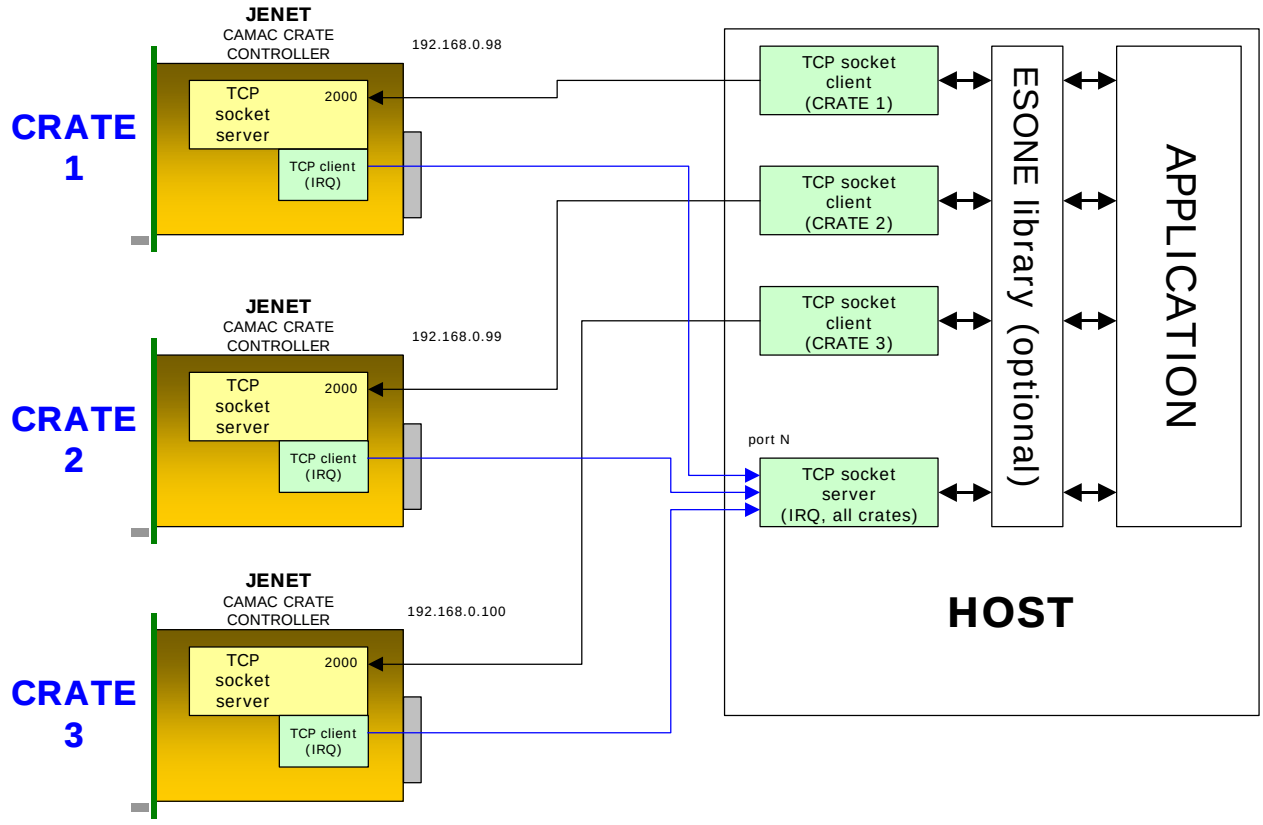
Remote control is organized as follows:

- the local firmware opens a TCP socket server on port 2000 and waits for connection from a remote client;
- when a client connection is detected, front panel LED status lights up; it will remain on until there are active connection to the socket server;
- the server accepts commands from the remote client (see section **Commands Reference**); return string is always in the following format:

| | |
|---|---|
| **-1** | Command exists, but parameter format or number is wrong |
| **-2** | Command does not exist |
| **0 <return value>** | Command returns correctly the return value (it may also be null) |

- note that more than one remote client can connect to the socket server; it is up to the programmer to avoid conflicts when accessing the same resources, as there is no built-in protection for access conflicts (in other words: stick to one remote client only unless you really know what your are doing). Multi-client usage is useful especially during development and debug.
- The socket server NEVER generates data autonomously; in the special case when Jenet needs to communicate to the host that a specific event has occurred (analogously to an interrupt request), it works through a separate communication channel.
- IRQ-like messages is handled in the following way: a TCP socket server is opened on the host at a specified port N. On Jenet, a TCP socket client is launched and connects to the IRQ socket server on host (port number N is passed with a dedicated message).
- The host must wait for data on the IRQ socket server and act accordingly.

In the following diagram, a typical scenario with 3 separate crates is shown.




A special command is represented by CNAME and deserves an additional explanation. CNAME stand for Connection Name and is a string identifier internally used by Jenet to understand who is talking to the local socket server. This feature is useful when multiple clients are connected and talking to the socket server. It is currently optional in firmware release 1.0, but we suggest using it at the beginning of connection to provide an upward compatibility for future releases.

| | |
|---|---|
| *Note:* | *On Windows 2000/XP, it is possible to perform a quick test with the Hyperterminal application, by specifying connection with TCP/IP and port 2000; on linux hosts, you can use the standard telnet client in "raw" mode , by typing*<br>`telnet <IP addr JNT01> 2000` |

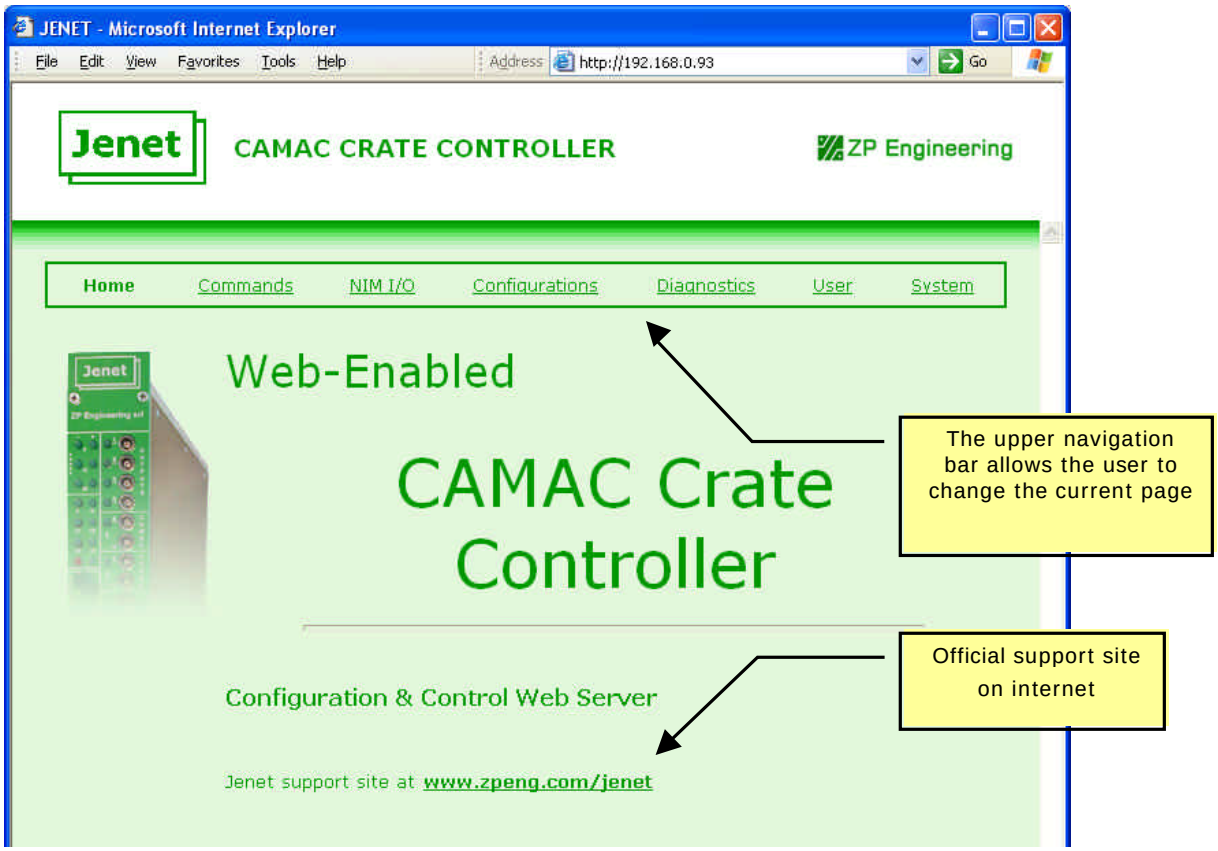
# *5.2 ESONE library*

The **ESONE library** is an ANSI C library, delivered in source form, providing an ESONE-like abstraction to the socket protocol, including multiple crate support. Documentation specific to the ESONE C library is on a separate document available on the Jenet support web site.

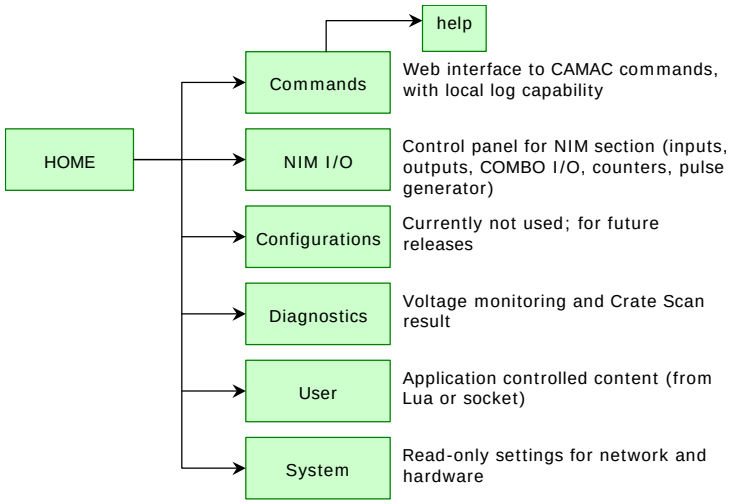
Note that the ESONE library is not mandatory; it is always possible to develop a library to directly handle communication at the socket level; however, the ESONE library facilitates migration of existing code and can be used as a reference for direct implementations.

# 6 Local web server

The local web server is a dynamic server, in the sense that gathers relevant data and information directly from the machine; therefore, depending on page contents, page refresh may be slower than expected if compared to a fully static web site.




***Note:*** *in order to trigger update operations in the optimal way, it is best to click on the link available on the navigation bar, instead of hitting the refresh button of the browser (i.e., F5 on Internet Explorer). This is due to the difference in HTTP requests that are sent by various web browsers when refreshing the page.*

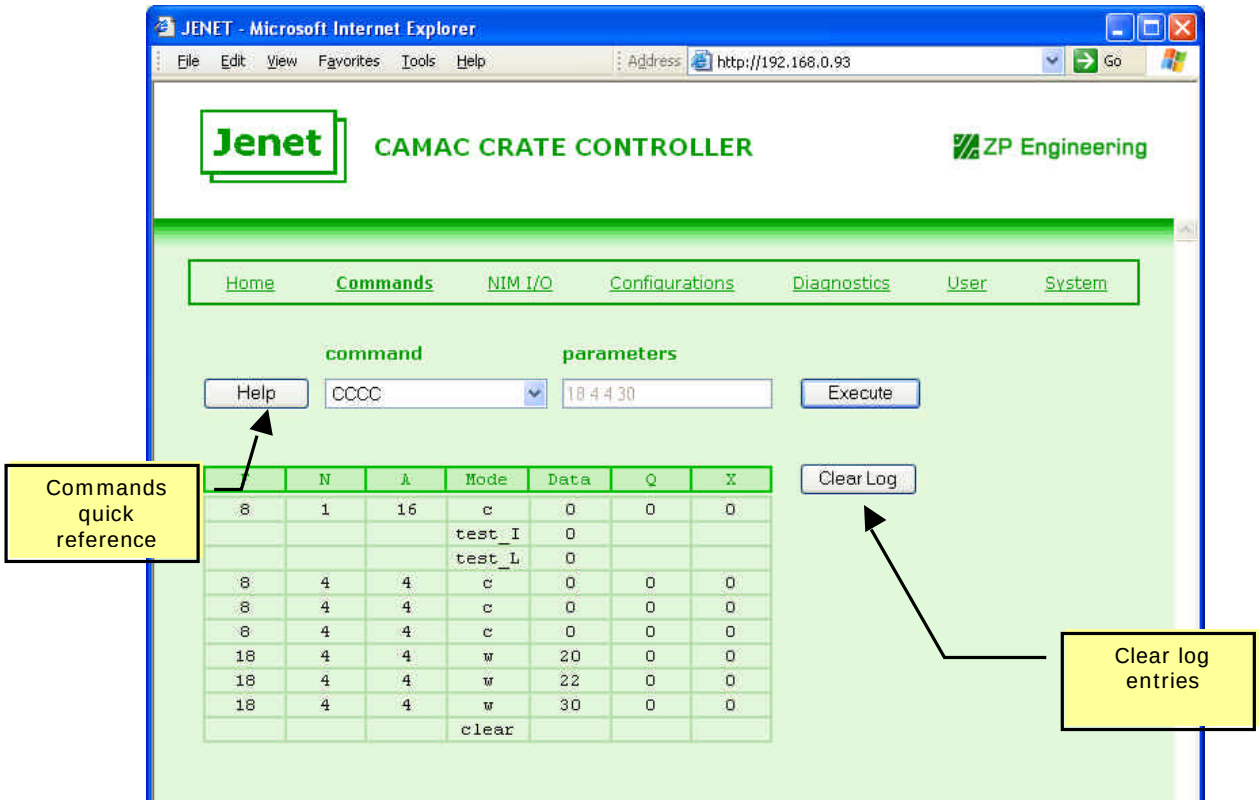


# 6.1 Commands page

The local web server contains a page dedicated to CAMAC commands, in order to allow immediate testing of crate functions. There is a log capability, currently limited to actions performed on the web page. A drop-down selection box allows the choice of one of 7 CAMAC commands; on the entry box the relevant parameters have to be entered; when pressing the EXECUTE button, the commands will be executed. For read functions, the result is available on the data field of the log section; for testing functions, the result is available in the data section, with values 0 or 1.

Please remember that in the current implementation logging is enabled only for commands executed from this web page. Logging is ten events deep; it is also possible to clear the log directly on the web page.

| Syntax definition | Description | Notes |
|---|---|---|
| **CSSA** <function> <slot> <subaddr> <data> | execute a CAMAC command with 16-bit data | response in Q<br>function=0..31, slot=1...23<br>subaddr=0...15 |
| **CFSA** <function> <slot> <subaddr> <data> | execute a CAMAC command with 24-bit data | response in Q<br>function=0..31, slot=1...23<br>subaddr=0...15 |
| **CCCC** | generate dataway initialize | |
| **CCCZ** | generate crate clear | |
| **CCCI** <value> | set/clear dataway inhibit | value=0 (reset), value=1 (set) |
| **CTCI** | test dataway inhibit | response in Q field |
| **CTLM** <which> | test LAM | response in Q field<br>which=1...23 |

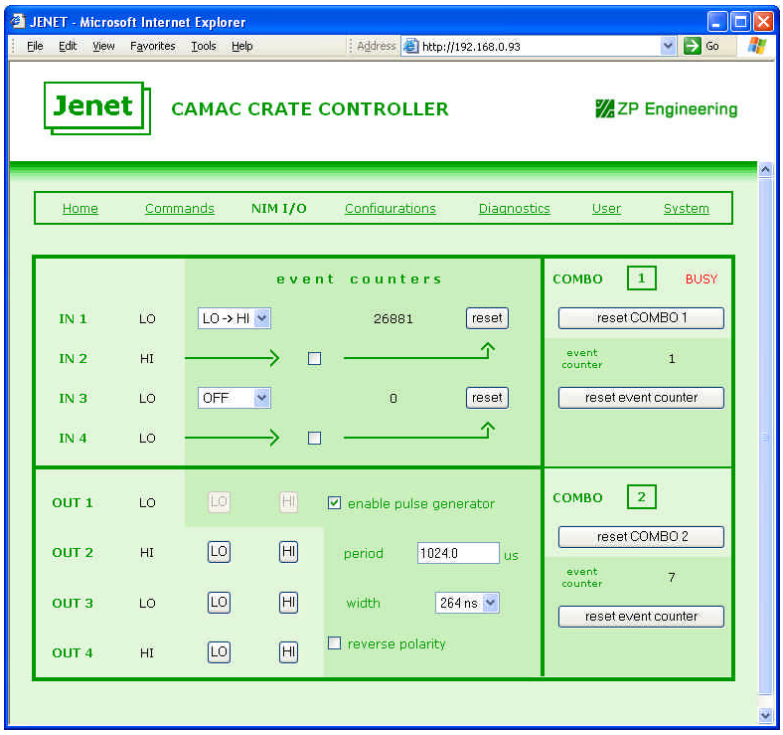


# *6.2 NIM I/O page*

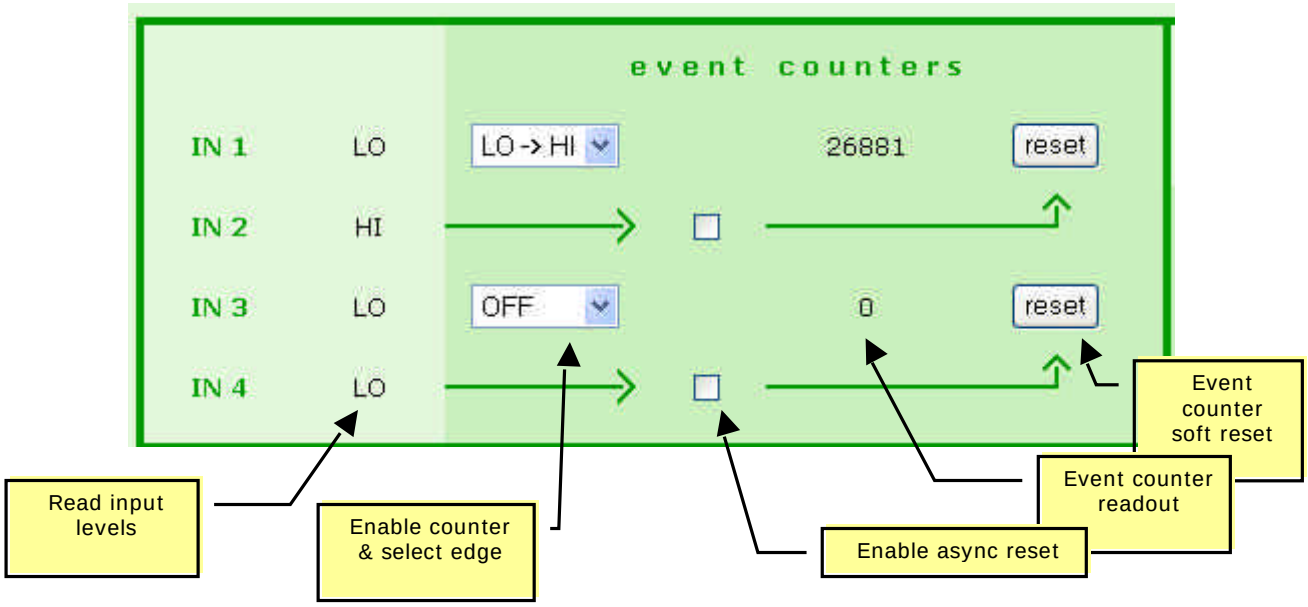
The NIM I/O page is arranged like a real control panel, in order to allow immediate interaction with the I/O section available on the front panel of the unit.

In addition, it is possible to retrieve default settings by pushing the DEFAULT front panel button (just above the RESET button). Default settings are stored with a specific socket command.

Note that not all interactions are allowed (a fuller control is available from socket or scripting), as browser access is considered unsafe from the remote control point of view.

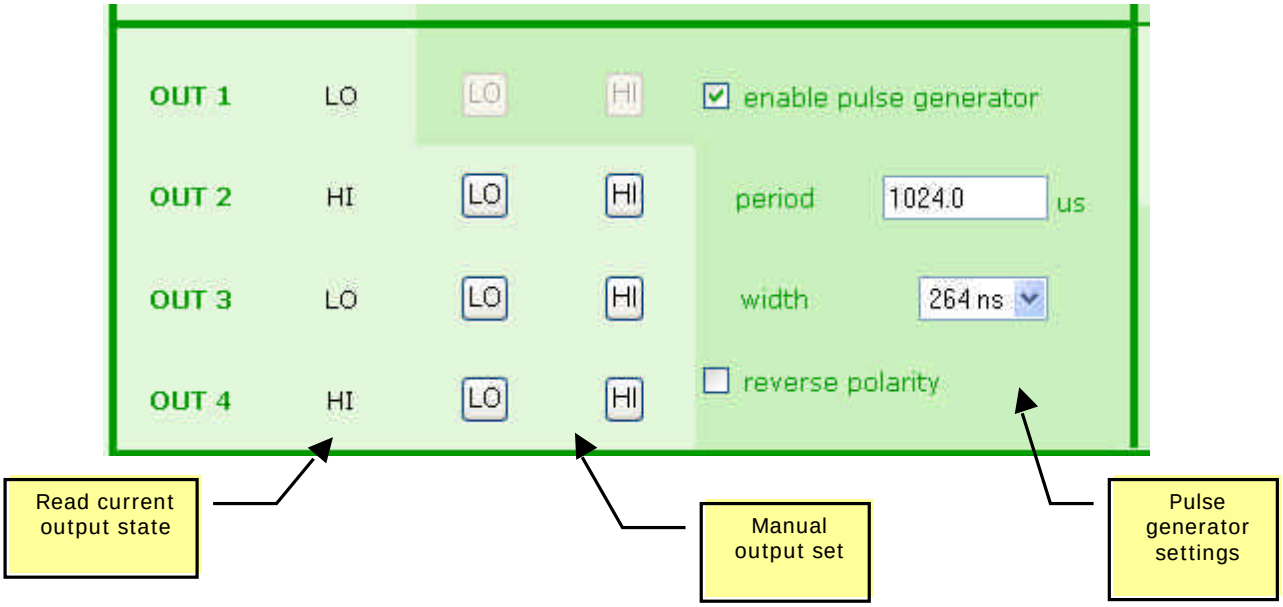



## 6.2.1 Input section



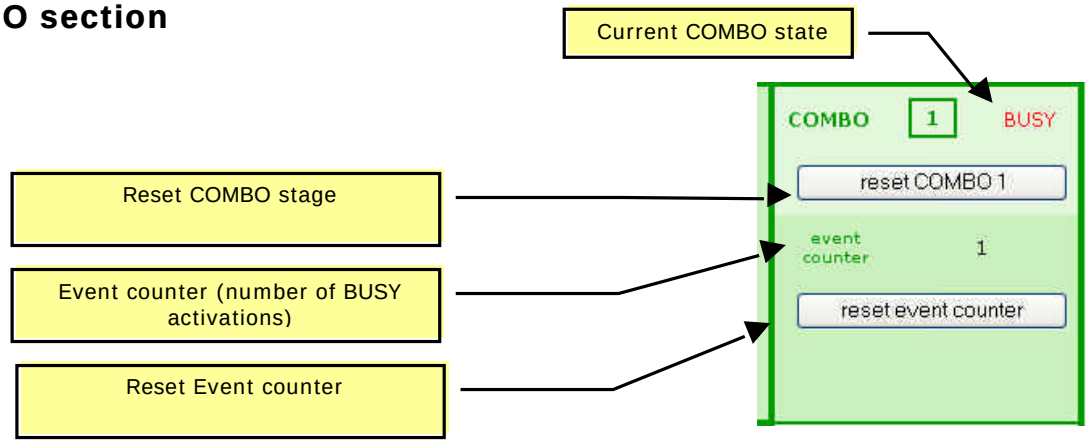

- two event counters can be enabled on inputs 1 and 3 independently, by selecting the required triggering transition (HI-to-LO or LO-to-HI) in the drop-down box
- event counter on input 1 can be asynchronously reset by input 2 (if the checkbox is flagged); there is also a button on the page that allows a software reset of the counter
- event counter on input 3 can be asynchronously reset by input 4 (if the checkbox is flagged); there is also a button on the page that allows a software reset of the counter

### 6.2.2 Output section



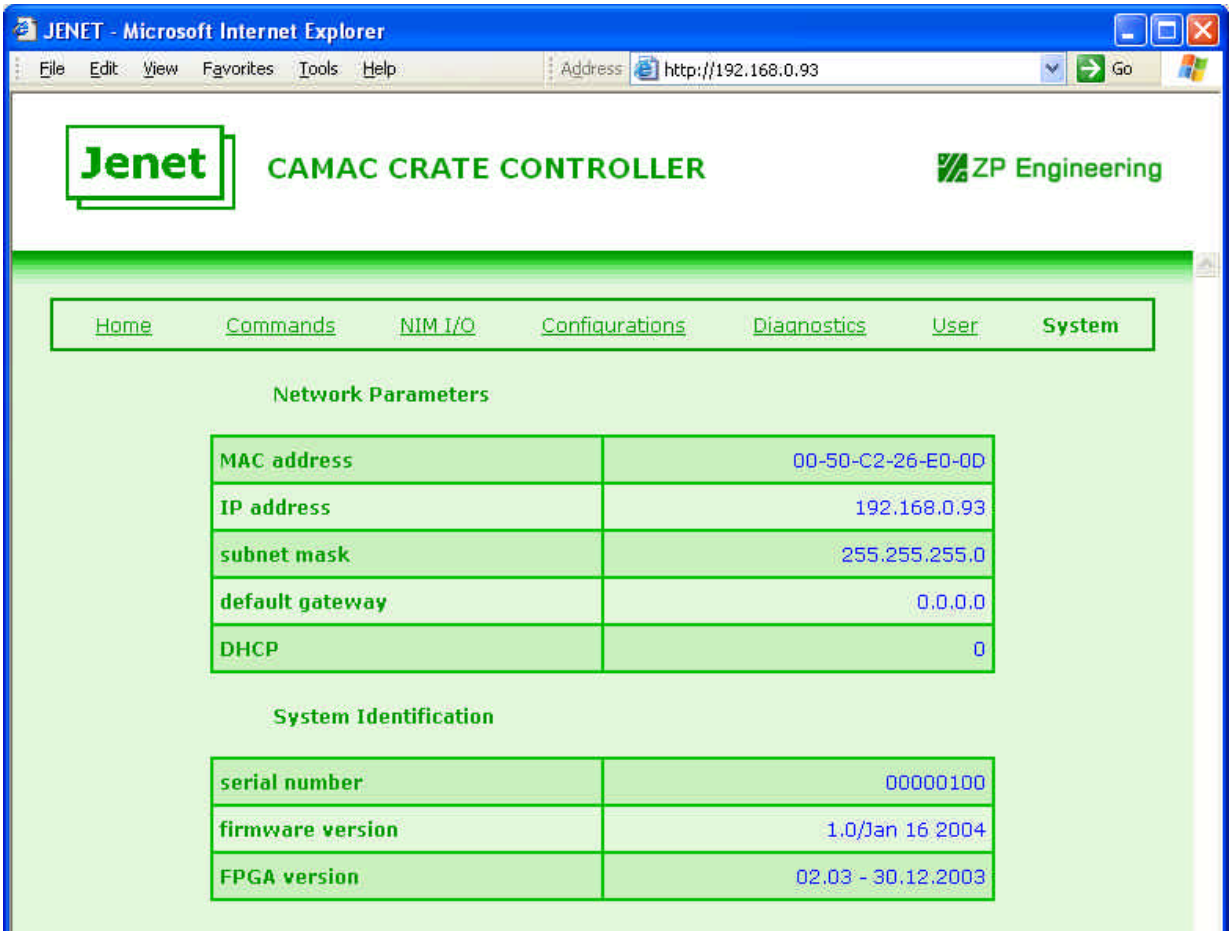

- output 1 can be set as a programmable pulse generator, for which the period, pulse width and polarity can be specified
- numerical entry of pulse generator period is automatically adjusted to fit the available resolution when the page is reloaded

### 6.2.3 COMBO section




- COMBO input 1 can be reset with the dedicated button; the usual way to reset a COMBO input would be from software
- COMBO input 2 can be reset with the dedicated button; the usual way to reset a COMBO input would be from software
- Dead Time Counter for both COMBO inputs is not shown on web page, as it has relevant meaning only immediately before resetting the COMBO input.

# *6.3 System settings page*



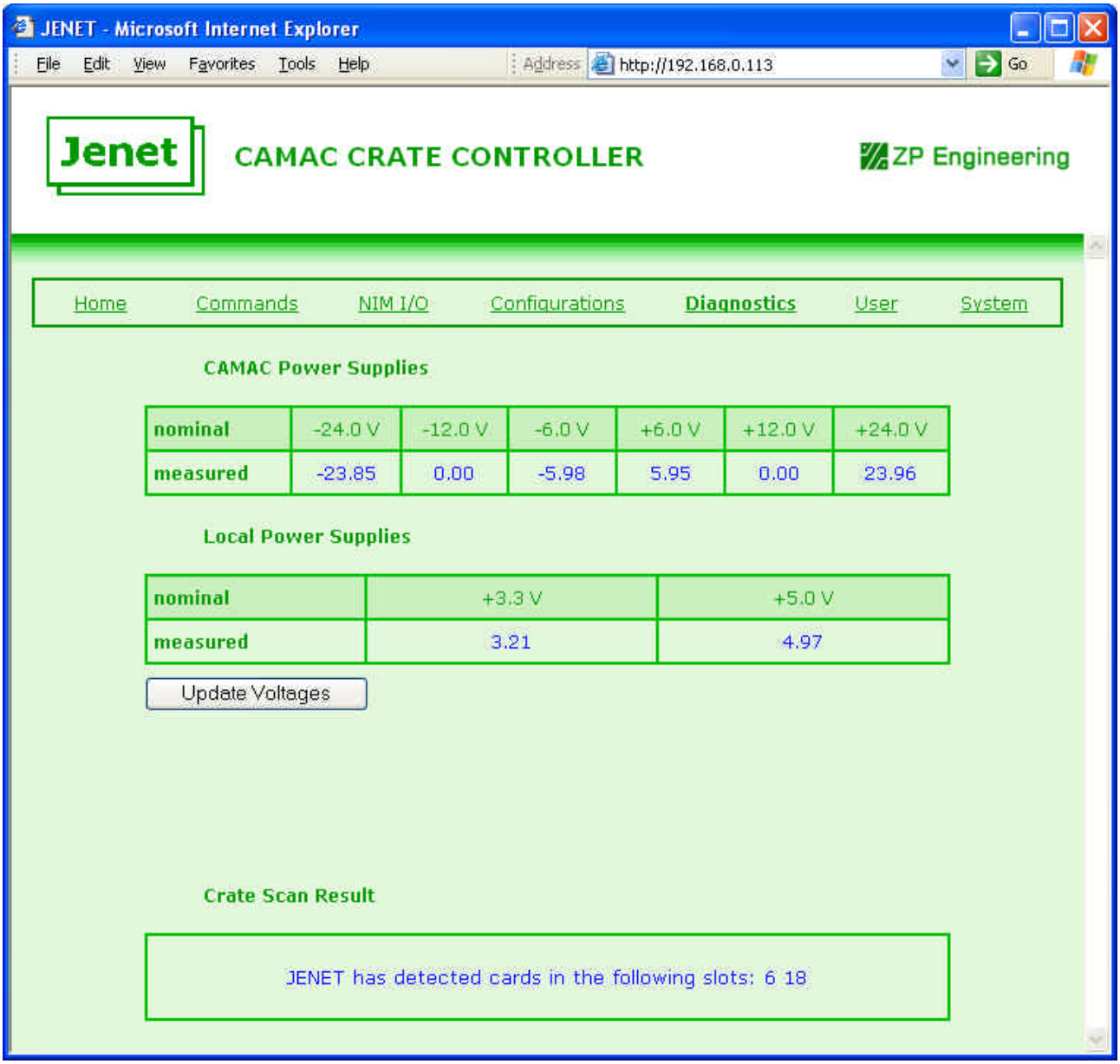

This is a read-only page that displays relevant network settings and local ID details; please always check that your documentation is referring to the same firmware version (release date is NOT relevant).

The MAC address is a read-only property that cannot be changed in any way, it may be useful in certain network environments; the local IP address, subnet mask, default gateway and DHCP enable flag can be modified from the serial port console (see **Serial Port Control** section) or form the control socket (take into account that these settings are effective only after rebooting). If DHCP = 1, then the local DHCP client is enabled. Please refer to your system administrator for additional information relevant to these settings.

Note that wrong or conflicting IP address settings are the most typical issue that creates connection problem when first using the system.

Before connecting a board with static IP (DHCP = 0), you can perform an additional check by issuing a PING command, to verify that the IP address on the unit is really available. If another system is answering at the same address, you may experience intermittent failure (i.e., the web page sometimes does not reload and all other communications will fail).

# *6.4 Diagnostics page*


JENET - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://192.168.0.113
Go
Jenet
CAMAC CRATE CONTROLLER
ZP Engineering
Home Commands NIM I/O Configurations Diagnostics User System
CAMAC Power Supplies
nominal -24.0 V -12.0 V -6.0 V +6.0 V +12.0 V +24.0 V
measured -23.85 0.00 -5.98 5.95 0.00 23.96
Local Power Supplies
nominal +3.3 V +5.0 V
measured 3.21 4.97
Update Voltages
Crate Scan Result
JENET has detected cards in the following slots: 6 18

This page shows current voltage readouts, divided into crate power supplies and local (digital-only) power supplies.

Note that Jenet requires the following voltages to be present in order to work properly:

-24V, -6V, +6V, -24V.

On the bottom of the page, results of the Crate Scan (performed only at power-up if the CSCAN flag is enabled, see **Startup Options** section) are shown in textual form. Remember that if Crate Scan is problematic (depending on which cards are inserted into the crate) it can be disabled either by serial or control socket.

# 7 Local scripting

## 7.1 Lua scripting language

An on-board script interpreter is available, allowing complex interactions with the CAMAC bus and local NIM I/O section. The scripting language is Lua, extended with a TCP/IP library and with specific commands that allow management of the underlying hardware.

Lua is a powerful, lightweight programming language designed for extending applications, frequently used as a general-purpose, stand-alone language. More information is available at www.lua.org.

Please take not that, while being free software, it is property of Tecgraf (http://www.tecgraf.puc-rio.br/), of which we acknowledge the excellent work.

A brief description of the scripting engine and its extensions follows.

| | | |
|---|---|---|
| Lua | Version 4.0 | This is the base scripting engine. It has been conceived as an efficient, compact add-on scripting library for various applications. |
| Luasocket library | Version 1.4 | Extension to Lua 4.0 (developed by Diego Nehab) that adds TCP and UDP functionality to the Lua scripting language. |
| Bit manipulation extension | -- | Added by ZP Engineering srl to offer bit-wise AND, OR, XOR functions |
| NIM I/O extension | -- | Added by ZP Engineering srl to offer full control of the local NIM I/O section |
| System extension | -- | Added by ZP Engineering srl to offer full control of system functions |
| CAMAC extension | -- | Added by ZP Engineering srl to offer full control over CAMAC commands and functions. |

Note that a complete reference of Lua is available on www.lua.org and on the Jenet support site (courtesy of the Lua community) at www.zpeng/jenet.

## 7.2 Lua engine in Jenet

One of the desirable features of Lua is that the language can be easily extended with new commands; this technique has been applied to allow full control of Jenet from a script.

A detailed reference of available extension commands is presented in chapter 9 (**Commands Reference**).

A dedicated application that connects to the control socket server allows the user to load the script and control its execution.

The script is launched and executed directly; in case of error, the FAULT red LED on the front panel is lit and script execution is halted. By issuing a **stop** command it is possible to restore the script engine to idle state.

The special function **doevents()** has been added to deal with the single-threaded nature of the scripting engine; it allows the system to terminate execution of the script itself. If this call is missing or called rarely, then it may not be possible to halt execution of the current script.

The ROB (Run-On-Boot) dedicated flag is available on the local EEPROM to indicate whether the FLASH script should be executed at startup. Note that there is no assumption of the temporal evolution coded into the script; therefore, many different uses may be made of the scripting capabilities.

### 7.2.1 Bit manipulation extension

As the base Lua library does not provide bit manipulation operands, ZP Engineering added some commands to allow binary operations, an essential feature for any embedded system. As an example, the CLMR Camac function returns the current LAM mask, and a binary AND operator is almost mandatory to allow individual bit testing.

The following additional functions are provided, all with two operands and one result:

| function | Description | C equivalent |
|---|---|---|
| band(a,b) | 32-bit binary AND | a & b |
| bor | 32-bit binary OR | a \| b |
| bxor() | 32-bit binary XOR (exclusive-OR) | a ^ b |
| bmod() | binary module | a % b |
| bsl | binary shift left | a << b |
| bsr | binary shift right | a >> b |

### 7.2.2 Socket commands for Lua control

A subset of control socket commands is dedicated to management of the local Lua interpreter engine; these commands are used by the jsm application (Jenet Script Manager, see below). Note that these socket commands are the only messages to violate the principle to have one command per line. The following commands are available:

| | |
|---|---|
| lua_setfile | It transfers a script from host to Jenet; the file is placed in RAM and can be saved on FLASH with the lua_store command. The following procedure is required:<br>• host sends to control socket the following command:<br>**lua_setfile <filesize>** where <filesize> is expressed in bytes<br>• control sockets answers with "OK"<br>• host sends the file directly<br>• control socket answers with "OK" |
| lua_getfile | It transfers a script from Jenet to host; the following procedure is required:<br>• host sends to control socket the following command:<br>**lua_getfile**<br>• control sockets answers with <filesize> (expressed in bytes)<br>• host sends "OK"<br>• control socket sends the file directly |
| lua_store | It saves the current script on FLASH |
| lua_getrun | It returns the current execution state of the script interpreter |
| lua_setrun | It changes the executionstate: lua_setrun <value>, where 1 = run, 0 = stop |
| lua_geterr | It returns the error message (if any) returned by the script interpreter; being a multi-line string, it follows the same protocol of lua_getfile |
| lua_getlog | It returns the stdout log file returned by the script interpreter; it's useful as a debugging aid as print() messages are sent to the log. Being a multi-line output, it follows the same protocol of lua_getfile.<br>WARNING: using print() on Lua may crash the system if the log file gets too long. Please use it only for debug !!!! |

### 7.2.3 Jenet Script Manager (jsm)

It is a dedicated application, available for Windows and Linux, that allows full control over all operations related to script management; all operations are specified with command line parameters. The following syntax is implemented (version 1.0):

| | |
|---|---|
| jsm -h | displays program version and a list of allowed parameters |
| jsm -ip <IP addr> -u <filename.ext> | uploads script text from specified file on host to Jenet |
| jsm -ip <IP addr> -run | starts execution of current script |
| jsm -ip <IP addr> -stop | halts execution of current script |
| jsm -ip <IP addr> -s | stores current script on non volatile memory for Run-on-Boot option |
| jsm -ip <IP addr> -rob <value> | sets rob (Run-on-Boot) flag to specified value (0 or 1) |
| jsm -ip <IP addr> -d <filename.ext> | downloads script text from Jenet to specified file on host |
| jsm -ip <IP addr> -d stdout | downloads script text from Jenet to stdout on host |
| jsm -ip <IP addr> -e <filename.ext> | stores error message (if any) from Jenet to specified file on host |
| jsm -ip <IP addr> -e stdout | stores error message (if any) from Jenet to stdout on host |
| jsm -ip <IP addr> -l <filename.ext> | stores log message (if any) from Jenet to specified file on host |
| jsm -ip <IP addr> -l stdout | stores log message (if any) from Jenet to stdout on host |

The jsm application is available in both source and compiled form on the documentation section of the Jenet web site (www.zpeng.com/jenet); it uses dedicated commands to transfer files. Please check periodically for updates.

### 7.2.4 Scripting on Jenet

A few usage examples follow.

**Script usage examples**

**Crate initialization**

In certain cases, it may be safe to initialize inserted CAMAC target cards as soon as possible after power-up.

With ROB = 1, the FLASH script performs the required initialization, terminating after completion

```
jn_led(1,1)          -- turn LED U1 on
run_once_init()      -- function somewhere else in the script
jn_led(1,0)          -- turn LED U1 off
```

**Automatic execution of monitoring loop**

After a run-once initialization section, an infinite loop is executed, calling as often as possible the doevents() function:

```
jn_led(1,1)          -- turn LED U1 on
run_once_init()      -- function somewhere else in the script
while (1) do
         doevents()
         run_in_loop() --  function somewhere else in the script
         pause (100)   -- wait 100 msec
end
```

**COMBO servicing**

```
jn_led(1,1)          -- turn LED U1 on
run_once_init()      -- function somewhere else in the script
while (1) do
         doevents()
         if (nim_testint(1) == 1) then
            do_something() -- function somewhere else in the script
            nim_cack(1)
         end
end
```

# 8 Firmware Upgrade

It is possible to upgrade a section of FLASH memory contents in order to allow firmware upgrades on the field, either to correct any problems that may show up during usage or to load application-specific executables.

Internal FLASH is structured into two banks: one for the operating system, and one for the application; the application bank includes all the executables related to Jenet.

| | |
|---|---|
| Note: | *When using a NFS-mapped disk, please remember to add the following line:*<br>*`<nfs_directory> <IP address JNT01>(rw,all_squash)`*<br>*to the **/etc/exports** file on the computer where disk is located.* |

Firmware upgrade (limited to the application bank) is performed using a telnet connection to the unit, as follows:

- copy the new binary file on a known location on a network disk
- establish a telnet connection on default port 23 (please note that, depending on fw version, it may be possible that the internal telnet server must be enabled by means of HW jumpers); of course, you must know the unit IP address
- type the following commands:

  ```
  cd www
  sh flash <nfs_directory> <jffs2 filename>
  ```

  and wait until reprogramming completes
- reboot the unit (either by pressing the RESET button on the front panel or by turning crate power off and on again).

An example of a typical fw upgrade command line is the following:

```
sh flash 192.168.0.91:/home/wet/bootdir jffs2.img
```

# 9 NIM subsection

Please note that on the present document the following notation is used for NIM signals:

| | |
|---|---|
| **LO** | No current flowing into load |
| **HI** | Current flowing into load (corresponding to a –0.8 V voltage level across a 50 ohm load) |

This notation is maintained also on the NIM I/O web page.

A command parameter equal to 1 means HI.

## *9.1 Default button*

A DEFAULT pushbutton, located on the front panel above the reset section, allows immediate manual reload of a default configuration for the NIM I/O subsection, thus allowing a quick reconfiguration of the system.

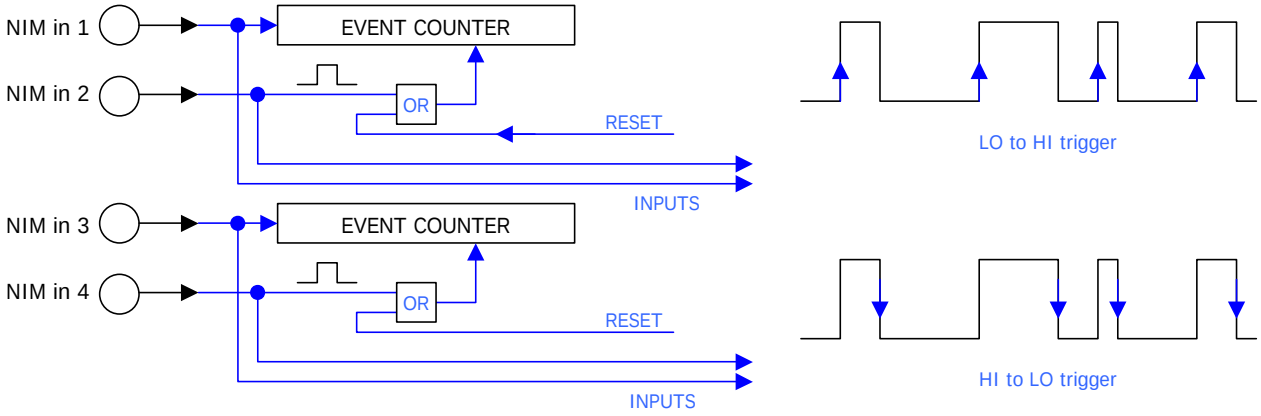
The same settings are applied at power-up, and can be set through the control socket.

The green LED located just below the DEFAULT pushbutton is turned on when default settings are applied, and turned off whenever any of the relevant settings is altered, providing an immediate visual feedback of the validity of default settings.

# *9.2 Inputs*

The NIM INPUT subsection implements four independent NIM inputs that may be read asynchronously to retrieve input status or configured to perform event counting; more specifically, inputs 1 and 3 can be set as event counter, triggering on rising or falling transition.

Counter reset is performed either with a dedicated control command or with an external reset, derived from input 2 (for counter on input 1) and from input 4 (for counter on input 3). Note that external reset capability must be enabled with the proper commands. External reset is active when relevant input is HI; while HI, counter is kept to zero and will not count further.

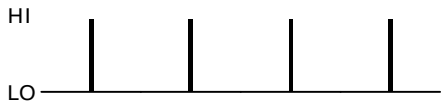
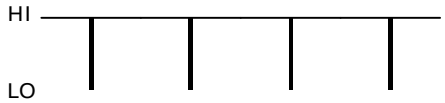



**EXAMPLES**

| **Read from inputs** | | **Notes** |
|---|---|---|
| TCP socket commands | Lua code snippet | Basic reading of input values (j=1,2,3,4) |
| `nim_getin`<br>`nim_getin j` | `A = nim_getin();`<br>`B = nim_getins(j)` | |
| **Set event counter on IN 3 to falling edge** | | |
| TCP socket commands | Lua code snippet | |
| `nim_setievcnt 3 1 1 0` | `nim_setievcnt(3,1,1,0)` | |
| **Enable async reset for counter on IN 1** | | |
| TCP socket commands | Lua code snippet | |
| `nim_setievcnt 1 1 1 0` | a,b,c=nim_getievcnt(1);<br>nim_setievcnt(1,a,b,1) | |

# *9.3 Outputs*

The OUTPUT subsection implements four independent NIM output that can be independently set and reset; in addition, a programmable pulse generator that can be enabled on the first output.

| **PULSE GENERATOR** | |
|---|---|
| Period | 186.2 µs to 95325.2 µs<br>(in 93.1 µs steps, 10791 Hz base clock) |
| Width | 88 ns to 616 ns |
| Output polarity | Direct   |
| | Reverse   |

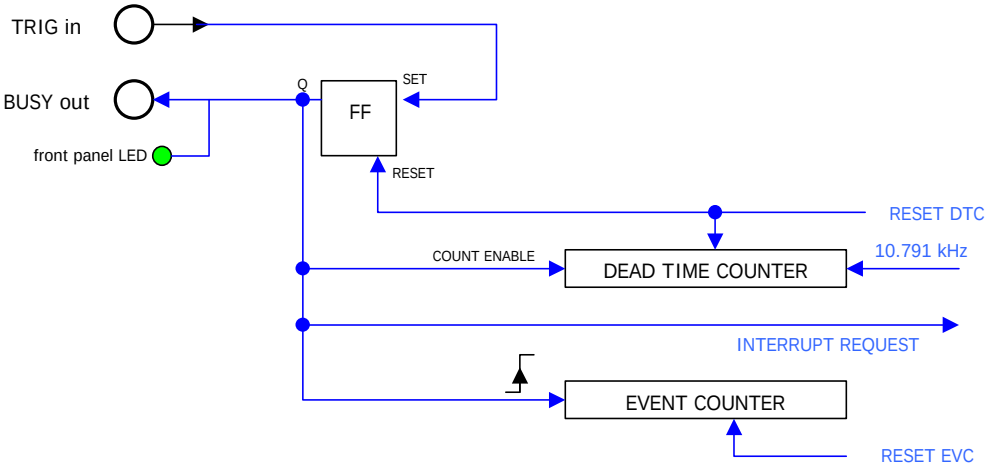
Please note that duty cycle is never higher than 0,05 % (with reversed polarity flag disabled), therefore it may be difficult in certain cases to see the pattern on a scope.

**EXAMPLES**

| **Write to outputs** | | **Notes** |
|---|---|---|
| TCP socket commands | Lua code snippet | Basic output control<br>out=1,2,3,4<br>val=0,1 |
| `nim_setout 1 1 1 1`<br>`nim_setouts <out> <val>` | `nim_setout(1,1,1,1)`<br>`nim_setouts(2,0)` | |
| **Set pulse generator to 1 ms period, 440 ns width** | | |
| TCP socket commands | Lua code snippet | 11*93.1 = 1,024 ms<br>5*88 = 440 ns |
| `nim_setpulse 11 5 0` | `nim_setpulse(11,5,0)` | |
| **Turn pulse generator off and activate outputs 1,3** | | |
| TCP socket commands | Lua code snippet | |
| `nim_pulseoff`<br>`nim_setouts 1 1`<br>`nim_setouts 3 1` | `nim_pulseoff()`<br>`nim_setouts(1,1)`<br>`nim_setouts(3,1)` | |

# 9.4 COMBO I/O

The COMBO I/O subsection implements a commonly requested Trigger/Busy functional block. The diagram below details one of the two COMBO subsections.




Whenever a LO-to-HI transition occurs on the TRIG input, an internal Flip-Flop is set to capture the event; the Flip-Flop output is connected to the BUSY output and its associated LED. The **Dead Time Counter** starts counting at a 10.791 kHz rate, and it is reset when the software application resets the Flip-Flop; therefore, the Dead Time Counter provides an indication of the service time of the TRIG input. It is automatically reset when BUSY is reset (therefore it must be read before the BUSY reset command).

An independent **Event Counter** is incremented at each LO-to-HI transition of the BUSY output, providing an indication of the number of pulses. The Event Counter can be reset by means of an explicit command (through control socket) or from the NIM I/O web page.

Please note that, in order to be ready to accept new events, the COMBO section must be explicitly served by the user, in one of the following ways: with a script section that resets the relevant COMBO FF; within the application running on host, with a routine that in response to the COMBO event resets the relevant COMBO FF; by clicking on the RESET button on the NIM I/O web page.

| **COMBO I/O** | |
|---|---|
| Event Counter | Incremented on BUSY output rising edge |
| Trigger event | TRIG input rising edge |
| Dead Time Counter | 1 (92.67 usec) to 65535 (6.7 sec) |
| TRIG-to-BUSY response | 35 ns |

## EXAMPLES

| **Acknowledge COMBO 1 trigger** | | **Notes** |
|---|---|---|
| TCP socket commands | Lua code snippet | |
| `nim_cack 1` | `nim_cack(1)` | |
| **Wait for TRIG on 1 and then turn NIM outs 3,4 ON** | | |
| TCP socket commands | Lua code snippet | |
| --- | nim_waitint(1); nim_cack(1)<br>jn_led(3,1); jn_led(4,1) | Blocking mode (wait) not available on socket |
| nim_testint 1 {repeat then}<br>nim_cack 1<br>jn_led 3 1<br>jn_led 4 1 | if (nim_waitint(1) == 1) then<br>  nim_cack(1); jn_led(3,1); jn_led(4,1)<br>end | Test mode |

# 10 Commands reference

A reference of available commands on both the TCP control socket server and the local Lua scripting language is detailed in this section.

Note that whenever possible, TCP control commands are as short as possible in order to minimize Ethernet traffic; on the other hand, Lua commands are more descriptive.

Some general rules follow:

- Control socket commands ALWAYS return an error code as first parameter; it is equal to 0 if the commands executed with success, -1 if command parameters are wrong, -2 if the command is non existent; in the table below this error code is NOT indicated when describing return values, as it must be always 0 to yield a correct answer.
- TCP control sockets are not case sensitive, while Lua commands are case sensitive (this is a specific property of the language).
- As much as possible, all commands have the same name for Lua and for TCP control socket.

| TCP control socket | Lua | Description |
|---|---|---|
| **CAMAC bus control** | | |
| `CCCZ` | `CCCZ()` | Generate Dataway Init |
| `CCCC` | `CCCC()` | Generate Crate Clear |
| `CFSA <function> <slot> <addr> <data>` | `CFSA(function,slot,addr,data)` | 24-bit CAMAC command; returns Q and data |
| `CSSA <function> <slot> <addr> <data>` | `CSSA(function,slot,addr,data)` | 16-bit CAMAC command; returns Q and data |
| `CCCI <value>` | `CCCI(value)` | Change Dataway Inhibit to specified value (0 or 1) |
| `CTCI` | `CTCI()` | CAMAC test Inhibit; returns 0 or 1 |
| `CTLM <slot>` | `CTLM(slot)` | CAMAC test LAM on specified slot = 1…..23 |
| `----` | `CCLWT(slot)` | CAMAC wait for LAM on specified slot (only for Lua) |
| `CLMR` | `CLMR()` | Returns current LAM register, in hex |
| `LACK <slot??>` | `LACK(slot??)` | LAM acknowledge |
| `CTSTAT` | `CTSTAT()` | Returns Q and X values (from last access on bus) |

| TCP control socket | Lua | Description |
|---|---|---|
| **DIAGNOSTIC SECTION** | | |
| vn24 | vn24() | Returns the measured voltage on –24V power supply, in float format |
| vn12 | vn12() | Returns the measured voltage on –12V power supply, in float format |
| vn6 | vn6() | Returns the measured voltage on –6V power supply, in float format |
| vp6 | vp6() | Returns the measured voltage on +6V power supply, in float format |
| vp12 | vp12() | Returns the measured voltage on +12V power supply, in float format |
| vp24 | vp24() | Returns the measured voltage on +24V power supply, in float format |
| vp33 | vp33() | Returns the measured voltage on +3.3V power supply, in float format |
| vp5 | vp5() | Returns the measured voltage on +5V power supply, in float format |
| **SYSTEM PARAMETERS (stored in EEPROM)** | | |
| ee_getdhcp | ee_getdhcp() | Returns 0 if DHCP client is not enabled, 1 otherwise |
| ee_getdns | ee_getdns() | Returns current DNS, in dotted notation |
| ee_getgw | ee_getgw() | Returns current Gateway, in dotted notation |
| ee_getip | ee_getip() | Returns current IP address, in dotted notation |
| ee_getmask | ee_getmask() | Returns current IP mask, in dotted notation |
| ee_getmac | ee_getmac() | Returns current MAC address, in hex format with '-' delimiters, like in "00-50-C2-26-E0-0A" |
| ee_setrob <value> | --- | Sets Run-on-Boot flag to specified value (0 or 1) |
| ee_getrob | --- | Returns current Run-on-Boot flag value (0 or 1) |
| ee_getserial | --- | Returns Jenet serial number |
| ee_setcscan | --- | Sets Crate Scan flag to specified value (0 or 1) |
| ee_getcscan | --- | Returns current Crate Scan flag value (0 or 1) |
| ee_storeconf | --- | Stores current NIM section configuration into EEPROM default settings |

| TCP control socket | Lua | Description |
|---|---|---|
| **NIM I/O SECTION** | | |
| nim_getin | nim_getin() | Returns NIM input values; it returns "a(1) a(2) a(3) a(4)", where a(n) = 0 or 1 |
| nim_getins <which> | nim_getins(which) | Returns a single NIM input value (0 or 1); <which> = 1, 2, 3 or 4 |
| nim_setievcnt <which> <en> <pol> <ext_reset> | nim_setievcnt(which,en,pol,ext_reset) | Sets Input Event Counter, with parameters <which><enable> <trig edge polarity> <external reset enable> |
| nim_getievcnt <which> | nim_getievcnt(which) | Returns Input Event Counter settings |
| nim_geticnt <which> | nim_geticnt(which) | Returns current Input Event Counter value |
| nim_reseticnt <which> | nim_reseticnt(which) | Reset specified Input Event Counter: <which> = 1 or 3 |
| nim_getouts <which> | nim_getouts(which) | Returns current value of specified output; <which> = 1…4 |
| nim_getout | nim_getout() | Returns current value of all outputs |
| nim_setouts <which> <value> | nim_setouts(which,value) | Sets value of specified output; <which> = 1…4 |
| nim_setout <v1> <v2> <v3> <v4> | nim_setout(v1,v2,v3,v4) | Sets current value of all output; <which> = 1…4 |
| nim_setpulse <period> <width> <polarity> | nim_setpulse(period,width,polarity) | Sets Pulse Generator, values in decimal (period = 1….1023, width=1…7, polarity= 0 or 1) |
| nim_getpulse | nim_getpulse() | Returns Pulse Generator settings |
| nim_pulseoff | nim_pulseoff() | Returns Pulse Generator settings |
| nim_setcthr <which> <value> | nim_setcthr(which, value) | Sets threshold value for COMBO section; <which> = 1 or 2 |
| nim_getcthr <which> | nim_getcthr(which) | Returns current threshold value for COMBO section; <which> = 1 or 2 |
| nim_getcdtc <which> | nim_getcdtc(which) | Returns current COMBO Dead Time Counter value; <which> = 1, 2 |
| nim_cack <which> | nim_cack(which) | Resets COMBO input; <which> = 1 or 2 |
| nim_getcev <which> | nim_getcev(which) | Returns current COMBO Event Counter value; <which> = 1, 2 |
| nim_resetcev <which> | nim_resetcev(which) | Reset specified COMBO Event Counter: <which> = 1 or 2 |
| nim_resetcombo <which> | nim_resetcombo(which) | Resets specified COMBO section; <which> = 1 or 2 |
| --- | nim_waitcombo(ch) | Wait for COMBO interrupt; it returns only when COMBO is busy (blocking call) ; ch = 1, 2 or 3 (= both) |
| --- | nim_waitdtc(ch) | Wait for DTC interrupt; it returns only when DTC has crossed the threshold (blocking call) ; ch = 1, 2 or 3 (= both) |
| nim_testcombo <ch> | nim_testcombo (ch) | Test COMBO interrupt; it immediately returns current values (1 if COMBO busy, 0 otherwise); ch = 1, 2 or 3 (= both) |
| nim_testdtc <ch> | nim_testdtc(ch) | Test DTC interrupt; it immediately returns current value (1 if DTC has crossed the threshold, 0 otherwise) ; ch = 1, 2 or 3 (= both) |

| TCP control socket | Lua | Description |
|---|---|---|
| **Control socket only commands** | | |
| cname <id> | -- | Optional command: it passes to Jenet an string identifier of local client socket. |
| lua_setfile <data..........> | -- | Lua-related command; enables direct transfer of script file |
| lua_store | -- | Lua-related command; stores transferred filew into non-volatile memory |
| lua_getfile | -- | Lua-related command; enables direct retrieval of script file |
| lua_getrun | -- | Lua-related command; returns Lua RUN flag |
| lua_setrun <value> | -- | Lua-related command; sets Lua RUN flag |
| lua_geterr | -- | Lua-related command; returns last Lua error message |
| lua_getlog | -- | Lua-related command; returns log file (in place of stdout) |
| **SYSTEM IDENTFICATION and VARIOUS** | | |
| ver | ---- | Returns a string containing firmware and FPGA versions |
| jn_fpgaver | jn_fpgaver() | Returns a string containing FPGA details (read-only information from FPGA registers) |
| jn_fwver | jn_fwver() | Returns a string containing the current firmware version running on the system |
| jn_initstat | ----- | Get results of board initialization; it should always be 0 |
| jn_led <which> <value> | jn_led(which,value) | Set front panels LEDs (U1, U2, U3, U4); <which> = 1 to 4, <value> = 0 or 1 |
| web_setuser <text> | web_setuser(text) | Sets text that appears in User page (see web server section) |
| web_getuser | web_getuser() | Returns text appearing in User page (see web server section) |
| **Lua only commands** | | |
| -- | doevents() | Enable system control of Lua loop |
| -- | pause(msec) | Wait for specified interval (in msec) |
| | band(a,b) bor(a,b)<br>bxor(a,b) bmod(a,b)<br>bsl(a,b) bsr(a,b) | Binary operators:<br>AND: a & b OR: a \| b<br>XOR: a ^ b MOD: a % b<br>BSL: a << b BSR: a >> b |

# 11 System Specifications

| Power Supply | |
|---|---|
| Required voltages on crate | +24 V (100 mA)<br>- 24 V (100 mA)<br>- 6 V (100 mA)<br>+ 6 V (700 mA) |
| Ethernet interface | |
| Settings | 10/100 Mbit autonegotiating |
| Default configuration | **IP address**: 192.168.0.98<br>**subnet mask**: 255.255.255.0<br>**default gateway**: 0.0.0.0<br>**DHCP client**: 0 (disabled)<br>(default configuration can be changed through serial port) |
| Serial port | |
| Settings | 38400 baud, 8-N-1, no flow control |
| Function | for configuration changes:<br>- system settings<br>- startup options |
| CAMAC bus interface | |
| Compliance | to ANSI-IEEE std. 583-1982 |
| NIM interface | |
| Connectors | SUHNER type |
| CPU section | |
| Processor | ARM7TDMI running at 44 MHz |
| Operating System | uClinux ver. 2.4.17 |
| FLASH | 4 Mbytes |
| SDRAM | 16 Mbytes |